APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: DETERMINING AN AMOUNT OF DATA READ FROM A STORAGE MEDIUM

APPLICANT: KNUT S. GRIMSRUD AND AMBER D. HUFFMAN

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EL245449758US

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.

March 31, 2000
Date of Deposit

Derek Norwood
Signature

Derek W. Norwood
Typed or Printed Name of Person Signing Certificate

DETERMINING AN AMOUNT OF DATA READ FROM A STORAGE MEDIUM

BACKGROUND

This invention relates to determining an amount of data that has been read from a storage medium while a data transfer is in progress.

When reading data from a storage medium, such as a hard disk, a device driver reads the requested data, called "demand data", optionally along with speculative data from other locations on the hard disk. The data from the other locations is called "prefetch data" and corresponds to addresses on the hard disk that are likely to be read next by the device driver (typically contiguous/sequential data).

DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of a disk drive and a host processing device.

Fig. 2 is a flow diagram showing a process for reading data from the disk drive.

Fig. 3 is a top view of a hard disk and a transducer head in the disk drive.

Fig. 4 is a front view of a computer which can function as the host processing device.

Fig. 5 is a block diagram of hardware included in the disk drive and the host processing device.

Like reference symbols in the various drawings indicate like elements.

DETAILED DESCRIPTION

Referring to Fig. 1, block diagram 10 shows a host processing device 11 reading data from a hard disk 12 of a disk drive 14. Disk drive 14 may reside within host processing device 11 or it may be an external drive.

Software (device driver) 15 executing in host processing device 11 receives requests from applications or other computer program(s) (not shown) executing on host processing device 11. These requests instruct device driver 15 to read data from locations on hard disk 12 of disk drive 14. Data is transferred from these locations back to a Direct Memory Access ("DMA") engine 16 also on host processing device 11. DMA engine 16 consults a table, such as scatter/gather list 26, to determine where in memory 17 to store the received data. Scatter/gather list 26 includes lists of addresses in memory 17 into which data from hard disk 12 is to be stored.

In Fig. 2, a process 19 is shown for reading data from hard disk 12. Device driver 15 receives (201) a request from software (a computer program) on host processing device 11. The request identifies an amount of demand data (the "demand request") at addresses of hard disk 12, and instructs device driver 15 to read the demand data from hard disk 12. Device driver determines the location and amount of prefetch data to

be read from hard disk 12 based on the location and/or amount of demand data and adds this information (the "prefetch request") to the original request.

Demand and prefetch data are typically, but not necessarily, contiguous data blocks. The demand data typically precedes the prefetch data (i.e., the prefetch data is retrieved after the demand data) in a direction of movement of hard disk 12. For example, as shown in Fig. 3, hard disk 12 rotates in a direction of arrow 21 during reading. A transducer head 22 on disk drive 14 reads data from tracks on hard disk 12 during rotation. Since the transducer head encounters demand data 24 before prefetch data 25 (as a result of the direction of rotation of hard disk 12), demand data 24 is read first. Prefetch data 25 is read on the assumption that the next data requested by host processor software will be the data that follows data 24.

Returning to Fig. 2, in response to the request received in 201, device driver 15 issuing read instructions (202) for demand data 24 and prefetch data 25 from hard disk 12. A program resident in a controller (not shown) on disk drive 14 provides the data to DMA engine 16 in response to the read instructions. DMA engine 16 then consults (203) a database, namely scatter/gather list 26, to determine where in memory 17 to store the data read in 202. Device driver 15 programs DMA engine 16 with scatter/gather list 26 prior to issuing the read instructions to hard disk 12. Scatter/gather list 26

4

includes entries which specify destination buffers (regions of memory 17) into which the demand data and the prefetch data are to be placed. The destination regions of memory may not be contiguous, in which case DMA engine 16 is programmed with multiple entries, one for each discontiguous region.

DMA engine 16 consults (203) scatter/gather list 26 periodically as blocks (associated with "LBAs", or "Logical Block Addresses") of data are read. The frequency at which scatter/gather list 26 is consulted may vary depending upon the size of the blocks of data.

Device driver 15 determines (204) the amount of data that has been read from hard disk 12 and satisfies a request based on the data. This may be done at any time during the reading process and it may be done in a number of ways. For example, DMA engine 16 may include a counter, which keeps track of the amount of data that has been read by counting the number of bytes transferred from hard disk 12 to host memory. Device driver 15 may monitor the current DMA transfer location via the counter to determine how much data has been stored in memory 17 since the start of the current reading operation and, thus, how much data has since been received from hard disk 12. Basically, device driver 15 checks DMA engine 16 to see how far along the DMA engine is in storing data from the current transfer operation.

Device driver 14 can determine (204) the amount of data that has been read before a request for demand and/or prefetch

data has been fully satisfied. Device driver 14 can then use this information to satisfy requests for data immediately, without waiting for all of the data to be read in response to the original request. For example, assume that a prefetch request has been issued for 64 kilobytes (KB) of data and is currently being satisfied (i.e., data is being read in response to the prefetch request). While the prefetch request is being satisfied, a demand request for 4 KB of data is received. These 4 KB constitute data that is part of the prefetch request. If the 4 KB of data have already been retrieved in response to the prefetch request, then the 4 KB of data can be transferred to the requestor (e.g., a software program) immediately to satisfy the request for 4 KB of demand data. Thus, there is no need to wait for the prefetch request for 64 KB of data to be completed before transferring the 4 KB of demand data. If the 4 KB of data have not yet been retrieved in response to the prefetch request, the 4 KB are transferred immediately when they are retrieved. This feature increases the operational efficiency of disk drive 10, since it eliminates the need to wait for an entire prefetch request to be completed before satisfying a demand request.

Another advantage associated with knowing how much data has been transferred at any given time relates to storing prefetch data. For example, sequential prefetch data may be read in response to a first request for data, where "sequential prefetch data" refers to data that follows demand

data in sequence (see data 25 of Fig. 3). If a second request is for non-sequential data, meaning data that does not follow the demand data in sequence, the prefetch request may be aborted and the prefetch data that has been read up to that point can be stored in memory 17. Since the amount of data that has been transferred is known, a record is maintained of the data that has been read and, thus, that data does not have to be re-read from hard disk 12 if it is needed. Heretofore, aborting a prefetch request in progress would result in such data typically being discarded, resulting in the need to re-read the data if that data was subsequently needed.

Accordingly, determining (204) may include the following. Device driver 15 determines (204a) if data that is currently being prefetched can be used to satisfy a demand request. In particular, device driver 15 consults DMA engine 16 to determine if the amount of data that has been read up to that point is sufficient to satisfy the demand request. If the amount of data is sufficient to satisfy the demand request, device driver 15 satisfies (204b) the request by transferring that data from a prefetch buffer portion of memory 17 to whatever computer program issued the original request. The data is transferred immediately without waiting for the prefetch request to be completed. If there is not sufficient data to satisfy the demand request, device driver 15 waits (204c) until sufficient data has been read and then satisfies (204c) the request as soon as that data has been read. Again,

7

it is not necessary to wait until the entire prefetch request has been satisfied. Representative pseudo-code for implementing this process is shown in the attached Appendix.

Hardware on which process 19 may be implemented is shown in Fig. 4. Personal computer ("PC") 32 includes disk drive 14 which reads and writes data on a hard disk, a display screen 34 which displays information, and input devices 35 which input data. A processor 36 (Fig. 5) in PC 32 runs device driver 15 and acts as the host processing device. DMA engine 16 uses scatter/gather list 26 (stored in memory 17).

Fig. 5 also shows components of disk drive 14. Among these components are hard disk 12, transducer head 22, pre-amplifier 37, analog variable gain amplifier ("VGA") 39, filter 40, analog-to-digital ("A/D") converter 41, processor 36 (including memory 17), and writer 45.

Hard disk 12 is a magnetic disk having concentric data storage channels defined on each of its storage surfaces. Hard disk 12 is rotated inside disk drive 14 while data is read from/written to its channels. Although only one hard disk 12 is shown, more than one disk may be included in disk drive 14.

Transducer head 22 is a magneto-resistive head (or similar device) that is capable of reading data from, and writing data to, hard disk 12. Transducer head 22 is associated in a "flying" relationship over a storage surface

of hard disk 12, meaning that it is movable relative to, and over, the storage surface in order to read and write data.

To read data from hard disk 12, device driver 15 (executing in processor 36) sends a signal to transducer head 22 to move transducer head 22 to locations on hard disk 12 from which data is to be read (process 19).

Transducer head 22 senses flux transitions as it moves in proximity to locations on hard disk 12. These flux transitions 50 are provided to pre-amplifier 37. Pre-amplifier 37 is a voltage pre-amplifier that amplifies the flux transitions from millivolts (mV) to volts (V). The resulting pre-amplified analog signal (or "read" signal) 51 is provided to VGA 39. VGA 39 amplifies read signal 51 and provides a resulting amplified read signal 52 to filter 40.

Filter 40 is a filter/equalizer that generates a substantially square wave from amplified read signal 52. Resulting filtered signal 54 is subjected to sampling and quantization within high-speed A/D converter 41. A/D converter 41 outputs digital data 55 generated from signal 54. Data 55 corresponds to the data stored on hard disk 12.

Writer 45 is provided for writing data to hard disk 12 (via transducer head 22). Memory 17 stores computer instructions (including software for device driver 15) for implementing process 19. Memory 17 also stores scatter/gather list 26.

Process 19 is not limited to use with the foregoing hardware and software configurations; it may find applicability in any computing or processing environment. Process 19 may be implemented in hardware, software, or a combination of the two. Process 19 may be implemented in one or more computer programs executing on programmable computers that each include a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and one or more output devices.

Each such program may be implemented in a high level procedural or object-oriented programming language to communicate with a computer system. However, the programs can be implemented in assembly or machine language. The language may be a compiled or an interpreted language.

Each computer program may be stored on a storage medium or device (e.g., hard disk or magnetic diskette) that is readable by a general or special purpose programmable computer for configuring and operating the computer when the storage medium or device is read by the computer to perform process 19. Process 19 may also be implemented as a computer-readable storage medium, configured with a computer program, where, upon execution, instructions in the computer program cause the computer to operate in accordance with process 19.

Other embodiments not specifically described herein are also within the scope of the following claims. For example,

process 19 can be used to access data stored on other storage media, including optical media, such as CDs ("Compact Disks"), DVDs ("Digital Video Disks"), and DLT ("Digital Linear Tape"). Device driver 15 may be an ATA ("Advanced Technology Attachment") driver. Process 19 may be executed in a different order from that shown and/or one or more blocks thereof may be executed concurrently. Additionally, while the invention has been described in the context of reading data from a storage medium, it can also be used in writing data to a storage medium.

APPENDIX

```
Procedure   PerformDiskRequest(DiskBlockNumber, Transfer Length)
{
      If (DiskisBusyWithPrefetch() && PrefetchLocation==DiskBlockNumber)
      {
            If (DMAProgress>=TransferLength)
                  //This is the optimized early demand completion
                  SatisfyRequestFromPrefetchBuffer();
                  Return;
            }
            else
            {
                  WaitForPrefetchToComplete();
                  //Alternatively, this could be:  while
                  //(DMAProgress<TransferLength)
                  SatisfyRequestFromPrefetchBuffer();
                  Return;
            }
      }
      else
      {
            WaitforDiskIdle();
            GetDemandDatafromDisk(DiskBlockNumber, TransferLength);
            StartPrefetchAtLocation(DiskBlockNumber + TransferLength);
      }
}
```